Parveen Khurana

124 Followers About Following

Perceptron Model

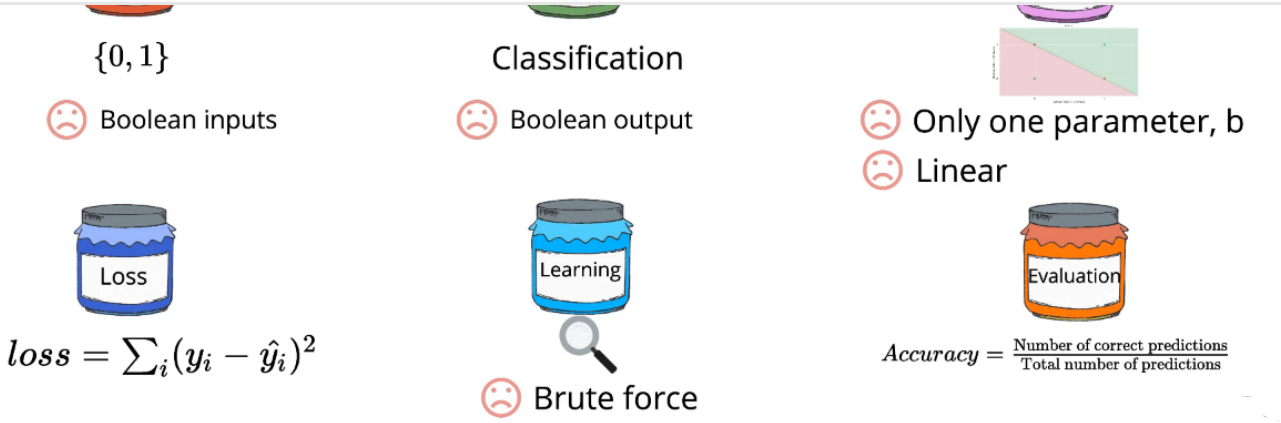
P Parveen Khurana Dec 2, 2019 · 16 min read

This article covers the content discussed in the Perceptron module of the Deep Learning course and all the images are taken from the same module.

In this article, we discuss the 6 jars of the Machine Learning with respect to the Perceptron model.

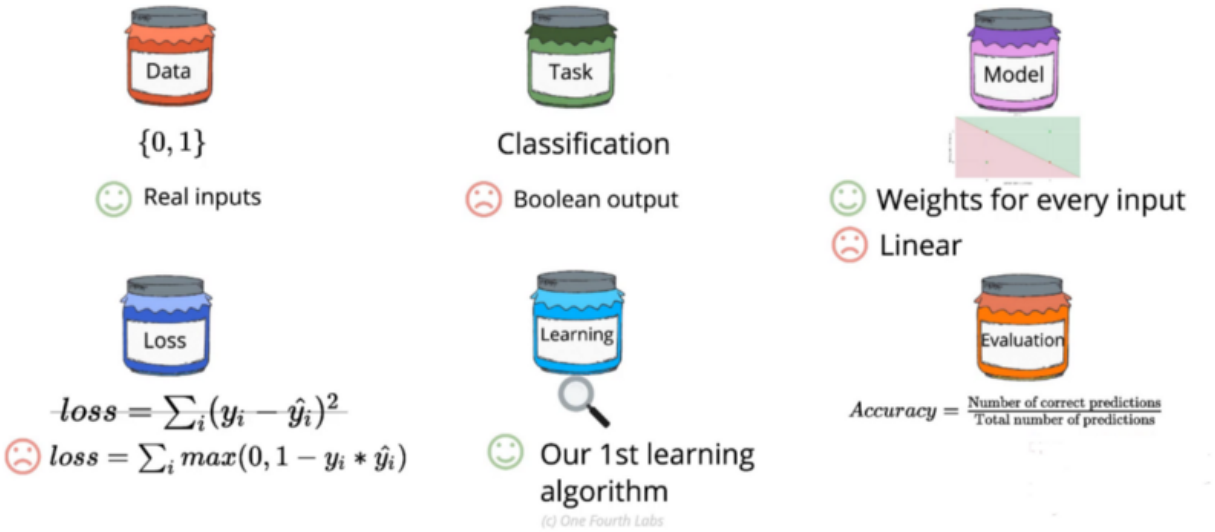
Our job in Machine Learning and in general in Deep Learning also is to find a function that captures the relationship between the input and the output. And this function has parameters(it could be the weights for the inputs, bias terms or some other parameters). So, our job is to come up with the function and its parameters using the data that we have.

Perceptron model tries to overcome the limitations of the MP Neuron model which are depicted below in terms of 6 jars of ML.



Limitations of MP Neuron with respect to 6 jars of ML

Perceptron model would overcome some of the limitations of the MP Neuron which we discuss in this article:



Perceptron Data Task:

When dealing with **MP Neuron**, the **data** that we could feed to the neuron **was all the Boolean data** and that lead to some unnatural decisions because, for example, in the real world we would like to have the absolute value of the weight instead of saying like it's heavy or light. Similarly, for screen size, we would like to deal with actual size instead of just saying small, medium or large. **It gives more flexibility to know the exact value.**

dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Data for MP Neuron

Below is the situation that we want

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Perceptron model can deal with Real value inputs and it could have **n** such inputs(**n features**), some of these can also be Boolean as per the requirement, but in general, we say it takes Real value input.

If we take a look at the above table, the price values are in orders of thousand(thousand of rupees) whereas the screen size is in orders of tens. So, **there is a difference in the range of these values**. The model will take all the features as the input and aggregate(be it weighted aggregate) and then take some decision based on

higher weight-age to one of these factors for example price might be more important point or screen size more important, that we could do that later on by adjusting the weights. But **to begin with, we would want any of the inputs to not have any unnatural advantage over the others in terms of the range of the values it can take. For example, if the price is say bringing in a value of 44000, that looks like a big number feed into the decision-making engine when some other numbers are going to be very small**. So, how does the model understand that this number is very big, I should scale down its importance(weight), which becomes difficult for the model to handle.

So, in all of the ML situations, we standardize the inputs.

Even we are dealing with Real numbers now, we still need to standardize the inputs, and that's where Data Preparation comes in.

Let's look at how to standardize screen size.

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	4.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (y)	1	0	1	0	1	1	0	1	0

screen size
5.8
6.18
5.84
6.2
5.9
6.26
4.7 min
6.41 max
5.5

So, for all the data points that we have, we can get the max screen size and the min screen size and we are going to standardize the data for each phone as per the below formula(for every phone, we re-compute the screen size which would be standardized)

Standardization formula

$$max - min$$

After the standardization, all the values lie in the range 0-1 with min value changed to 0 and the max value changed to 1.

screen size	screen size
5.8	0.64
6.18	0.87
5.84	0.67
6.2	0.88
5.9	0.7
6.26	0.91
4.7 min	0
6.41 max	1
5.5	0.47

In the same way, we would standardize the data for Battery

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	138	185	170
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	1960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (y)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

$$x' = \frac{x - min}{max - min}$$

battery	battery
3060	0.36
3500	0.51
3060	0.36
5000 max	1
3000	0.34
4000	0.67
1960 min	0
3700	0.57
3260	0.43

distance from 0 or 1, we know say 0.67 is actually a high value and 0.36 is a lower value. So, that difference is still retained, it's just the scale that reduces.

In the same way, we could standardize the data for all the features:

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

So, **Perceptron gives the flexibility to have real value inputs but to be able to deal with them in practice, the first thing that we should do is to standardize the input. The output is still going to be Boolean in the Perceptron model as well.**

So, the task that we can deal with is still a Binary Classification.

Perceptron Model:

The perceptron looks like the below image:

It looks like the MP Neuron model, **the key difference here** is that the **inputs are now going to be real values**, and **we also have a weight associated with each of these inputs**(all these weights are the parameters of the model and if we take all these weights as 1, then it is the same as the MP Neuron model). The actual function form of this model is as below:

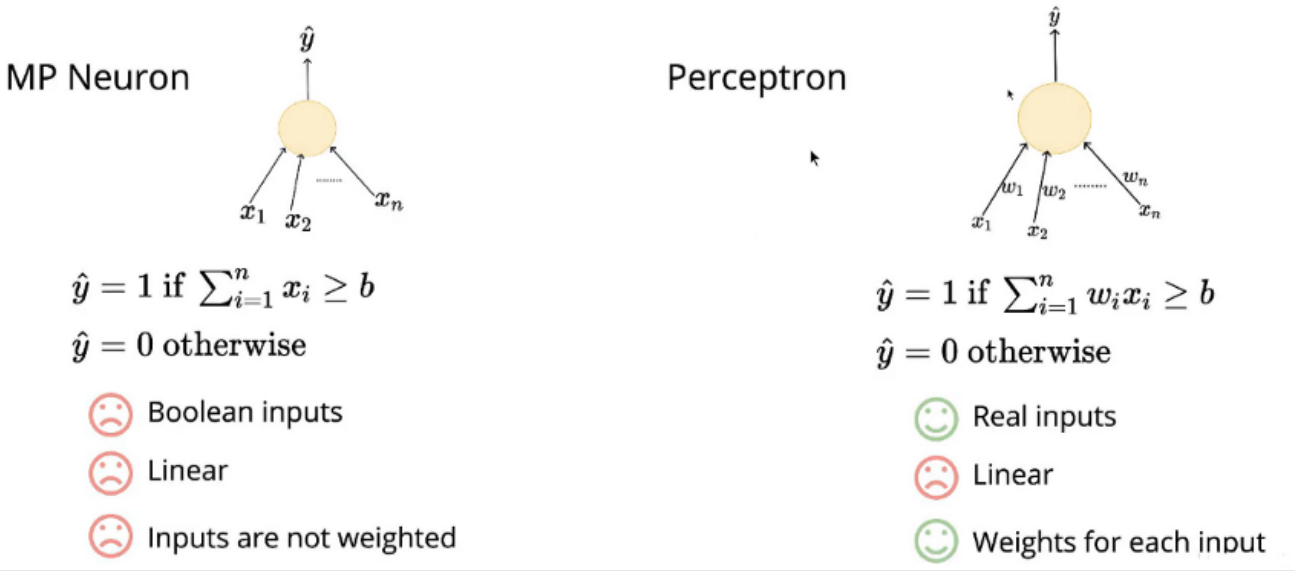
$$\hat{y} = 1 \text{ if } \sum_{i=1}^{n} w_i x_i \geq b$$

$$\hat{y} = 0 \text{ otherwise}$$

Again, this is very similar to the MP Neuron function. It can be represented as an if-else condition where the **output would be 1 if the weighted sum of the inputs is greater than equal to a threshold and the output is going to be 0 if the weighted sum is less than the threshold value**.

This function equation still looks like a straight line, if we expand it out for 3 dimensions, we have

w1x1 + w2x2 + w3x3 - b = 0



So, now the question is **why do we need weights?**

Let's took a simple case. Typically, we see that the likelihood to buy the phone might be inversely proportioned to the price.

$$\text{Like } \alpha \frac{1}{price}$$

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (g)	151	180	160	205	162	182	158	185	170
Screen size (inches)	5.8	6.18	5.84	6.2	5.9	6.26	5.7	6.41	5.5
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(mAh)	3060	3500	3060	5000	3000	4000	2960	3700	3260
Price (INR)	15k	32k	25k	18k	14k	12k	35k	42k	44k
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Now one of the input features in the above image would be the price of the phone, and if that price is very high, we want the output to be 0.

And we are taking the sum of all the inputs, let's say we have two inputs, in that case, the sum would be

w1x1 + w2x2

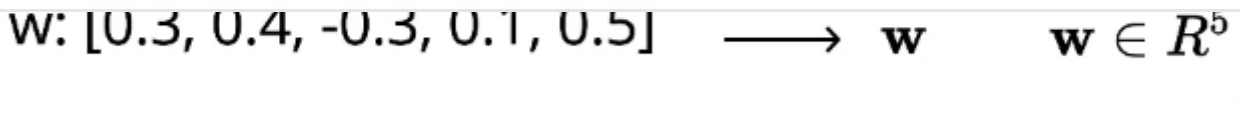
Say x2 is price, the higher the price, the lower is the chance to buy the phone, so that means, as the price increases, we want that the sum(w1x1 + w2x2) to not exceed the threshold. But as we are taking the summation so if the price increases the summation would also increase unless the weight associated with it(w2 in this case) is negative. So, if weight is -ve, then higher the price, the lower the entire summation would be. So, that means the summation would cross the threshold for some low prices phones but it will not cross the threshold as the price increases because the sum would become smaller and it might not be able to cross the threshold. So, that's the intuition behind having weights.

And it might be the case that higher the screen size, the more is the probability of buying a phone, so, in this case, we would like to assign a higher +ve weight with the screen size.

So, weights help to decide the importance of a feature and we could also assign negative weight-age to some feature.

All the features of a phone, we can represent as a vector(**X**) and each of the elements of the vector we can refer to by **x1, x2, ….., xn**. And in general, we can call them as **xi** for the **i'th input**.

For each feature, we are going to have a weight, so **for n features**, we would have **n weights** and this we could represent by a weight vector. And we could refer to the **i'th weight as wi**.



Now in the model function, we are doing the summation of the element-wise product(dot product) of these two vectors.

$$\mathbf{x}.\mathbf{w} = ?$$

$$\mathbf{x}.\mathbf{w} = x_1.w_1 + x_2.w_2 + ...x_n.w_n \quad = \sum_{i=1}^{n} x_i.w_i$$

$$\boxed{\mathbf{x}.\mathbf{w} = \sum_{i=1}^{n} x_i.w_i}$$

$$\hat{y} = 1 \ (\text{if } \mathbf{x}.\mathbf{w} \geq b)$$

$$\hat{y} = 0 \ (\text{otherwise})$$

So, we could say that the model would output 1 if the dot product of two vectors(input vector and the weight vector) is greater than some threshold.

Perceptron Loss Function:

Suppose we are making a decision based on two inputs

Weight	Screen size	Like (y)
0.19	0.64	1
0.63	0.87	1
0.33	0.67	0
1	0.88	0

$$L = 0, \text{ if } y = \hat{y}$$
$$= 1, otherwise$$

The loss would be 0 when the model's output is the same as the true output and we assign the model a penalty of 1 if the model's output is different from the true output.

The above can be represented as below(using an **indicator variable**)

$$L = \mathbf{1}_{(y \neq \hat{y})}$$

An indicator variable is denoted by 1 and it will have some condition associated with it(in subscript), what it means is whenever the condition is true, the indicator variable would take on a value of 1 and whenever this condition is false, then the indicator variable would take on a value of 0.

Suppose the predicted output is as below and the corresponding loss value has been computed accordingly

Weight	Screen size	Like (y)	$\hat{y}$	Loss
0.19	0.64	1	1	0
0.63	0.87	1	0	1
0.33	0.67	0	1	1
1	0.88	0	0	0

loss function ?

A. To tell the model that some correction needs to be done!

And the correction means the correction in the parameters of the model to be in the weights or threshold such that the overall loss is reduced.

Let's see how this loss function is different from the squared error loss function:

$$\text{Perceptron loss} = \mathbf{1}_{(y \neq \hat{y})}$$

$$\text{Squared Error loss} = (y - \hat{y})^2$$

Weight	Screen size	Like(y)	$\hat{y}$	Perceptron Loss	Squared Error Loss
0.19	0.64	1	1	0	0
0.63	0.87	1	0	1	1
0.33	0.67	0	1	1	1
1	0.88	0	0	0	0

In the above case, both the loss function values are exactly the same.

When the true output is not the same as the predicted output, then Perceptron Loss would be 1 and the squared error loss would also be 1

And when the true output is the same as the predicted output, both the loss values would be 0.

So, in the simple case where the outputs are Boolean, the Perceptron loss is similar to the squared error loss.

Perceptron Model Learning Algorithm:

Learning Algorithm is required to learn the parameters of the model using the data and the loss function.

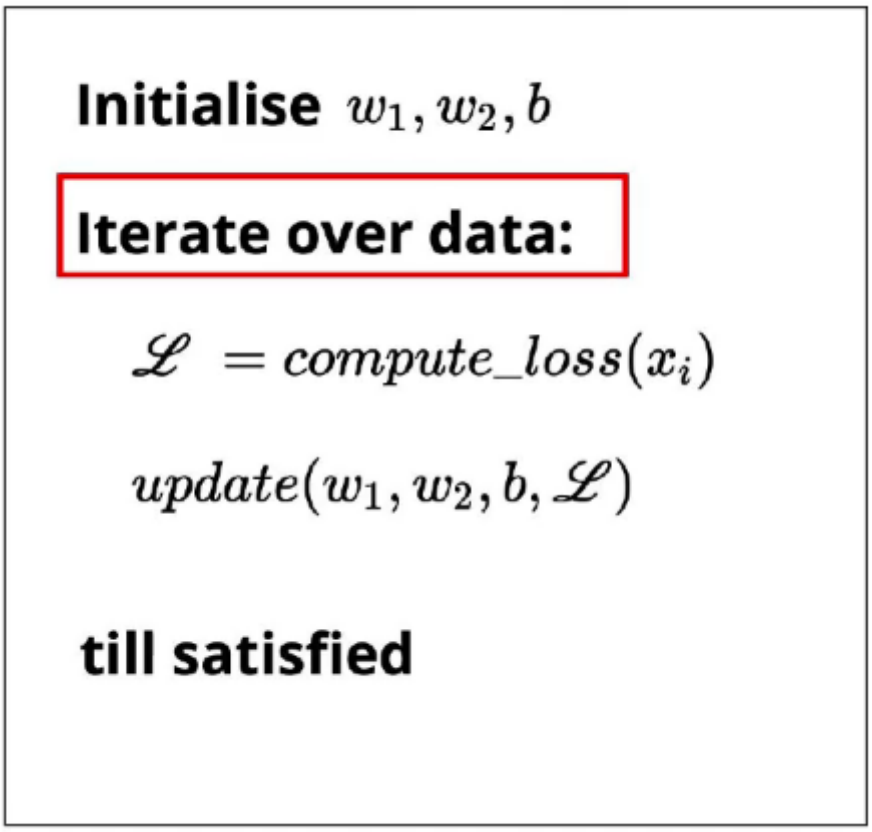
General Recipe for learning the parameters of the model:

Weight	Screen size	Like
0.19	0.64	1
0.63	0.87	1
0.33	0.67	0
1	0.88	0

Here, **in this case, we have the parameters w1, w2, and b.**

w1 corresponds to x1, w2 corresponds to x2 and b is the threshold.

the equation, get the output, compare it with true output and calculate the loss) and based on this loss value, we take an action and update the parameters and then we keep iterating over the data, go to the next point again compute the loss again update the parameters, then go to the next point and so on. Once we have gone through all the data points, we expect to be a little closer(overall Loss would be reduced) to the True output. And we keep repeating this(going over the data again and again) till we are satisfied with the loss value or the accuracy of the model.



Weight	Screen size	Like
0.19	0.64	1
0.63	0.87	1
0.33	0.67	0
1	0.88	0

Perceptron model is defined as

$$\hat{y} = \left(\sum_{i=1} w_i x_i \geq b \right)$$

And for 2 dimensions, we could write it as

$$w_2 x_2 + w_1 x_1 - b \geq 0$$

If **we define x0 as 1 and w0 as -b**, then we have

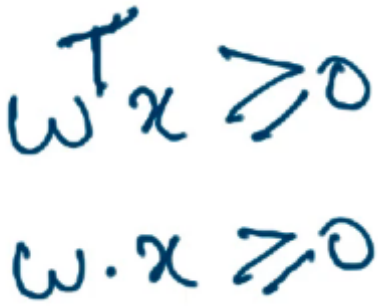
$$w_0 = -b$$
$$x_0 = 1$$

$$w_2 x_2 + w_1 x_1 + w_0 x_0 \geq 0$$
$$\downarrow$$
$$(-b)\ (1)$$

which we can write in compact form as

$$\frac{n}{}$$

So, instead of b, we have 0 now(in the equation on RHS) and the index(to iterate) from 0 instead of 1. And we can write this in the form of dot product as below:



$$\hat{y} = 1 \text{ (if } \sum_{i=0}^{n} w_i x_i \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$

$$\hat{y} = 1 \text{ (if } \mathbf{w}.\mathbf{x} \geq 0)$$
$$\hat{y} = 0 \text{ (otherwise)}$$

Algorithm: Perceptron Learning Algorithm

P ← *inputs with label* 1;
N ← *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**

```
    end
    if x ∈ N  and  Σ_{i=0}^{n} w_i * x_i ≥ 0 then
        w = w − x;
    end
end
//the algorithm converges when all the inputs are
  classified correctly
```

The intuition behind the Perceptron Learning Algorithm(updating the parameters):

We can represent the weights vector as(ignoring the bias and w0 for now):

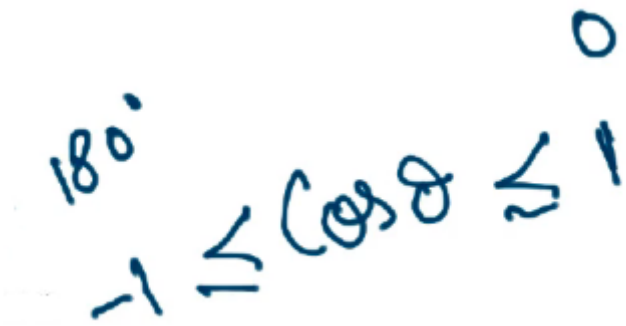
$$W = [w_1 \; w_2 \; .. \; w_n]$$

And the input features vector as:

$$x = [x_1 \; x_2 \; .. \; x_n]$$

Now the cosine of the angle between the vectors w and x is given by

$$\cos\theta = \frac{W \cdot x \leftarrow \sum w_i x_i}{\|w\| \, \|x\|}$$



So, we can say that whenever the cosine of an angle is between 0 and 1, the angle between two vectors would be an acute angle(from 0 degrees to 90 degrees) and whenever the cosine of an angle is less than 0, then the angle would be an obtuse angle(from 90 to 180 degrees).

So, the sign of cosine in the below expression would depend only on the dot product of the vectors w and x as the denominator would always be +ve.

$$\cos\theta = \frac{w \cdot x \leftarrow \sum w_i x_i}{\|w\| \|x\|}$$

So, if **w.x is ≥ 0**, then **θ** would be acute and lie between 0 to 90 degrees.

Algorithm: Perceptron Learning Algorithm

P ← *inputs with label* 1;
N ← *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**

```
    | w = w + x;
  end
  if x ∈ N and Σ_{i=0}^{n} w_i * x_i ≥ 0 then
    | w = w − x;
  end
end
//the algorithm converges when all the inputs are
 classified correctly
```

For positive points(True output as 1), **if w.x is -ve that means the angle between the vectors w and x lies between 90 to 180 degrees, but we want w.x to be ≥ 0(point lies on or above the line), or in other words, we want the angle to lie between 0 and 90 degrees.**

For $\mathbf{x} \in P$ if $\mathbf{w}.\mathbf{x} < 0$ then it means that the angle (α) between this $\mathbf{x}$ and the current $\mathbf{w}$ is greater than 90° (but we want α to be less than 90°)

The update rule for this scenario(highlighted below) would make sense only if the angle between the new value of w and x is actually lesser than what it was currently

```
while !convergence do
  Pick random x ∈ P ∪ N;
  if x ∈ P and Σ_{i=0}^{n} w_i * x_i < 0 then
    | w = w + x;
  end
  if x ∈ N and Σ_{i=0}^{n} w_i * x_i ≥ 0 then
    | w = w − x;
  end
end
```

What happens to the new angle ( ) when

$$\cos(\alpha_{new}) \propto \mathbf{W_{new}}^T \mathbf{X}$$
$$\propto (\mathbf{W} + \mathbf{X})^T \mathbf{X}$$
$$\propto \mathbf{W}^T \mathbf{X} + \mathbf{X}^T \mathbf{X}$$

Now the quantity(below) is always going to be positive(as it would just be the sum of the squares of its elements)

$$\mathbf{X}^T \mathbf{X}$$

So, we have the situation as

$$\cos(\alpha_{new}) \propto \mathbf{W_{new}}^T \mathbf{X}$$
$$\propto (\mathbf{W} + \mathbf{X})^T \mathbf{X}$$
$$\propto \mathbf{W}^T \mathbf{X} + \mathbf{X}^T \mathbf{X}$$
$$\propto \cos\alpha + \mathbf{X}^T \mathbf{X}$$

cosine of the new angle equals the cosine of the current angle plus some positive quantity that means the cosine is going to increase as compared to the cosine of the current angle between two vectors. So, if cosine is going to increase that means the angle is going to reduce

So, we can be assured that with this update(in the value of **w** as in Learning Algorithm), the angle between the weights vector and the input vector is going to reduce.

We can make a similar argument for the negative case.

while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\sum_{i=0}^{n} w_i * x_i < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\sum_{i=0}^{n} w_i * x_i \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end

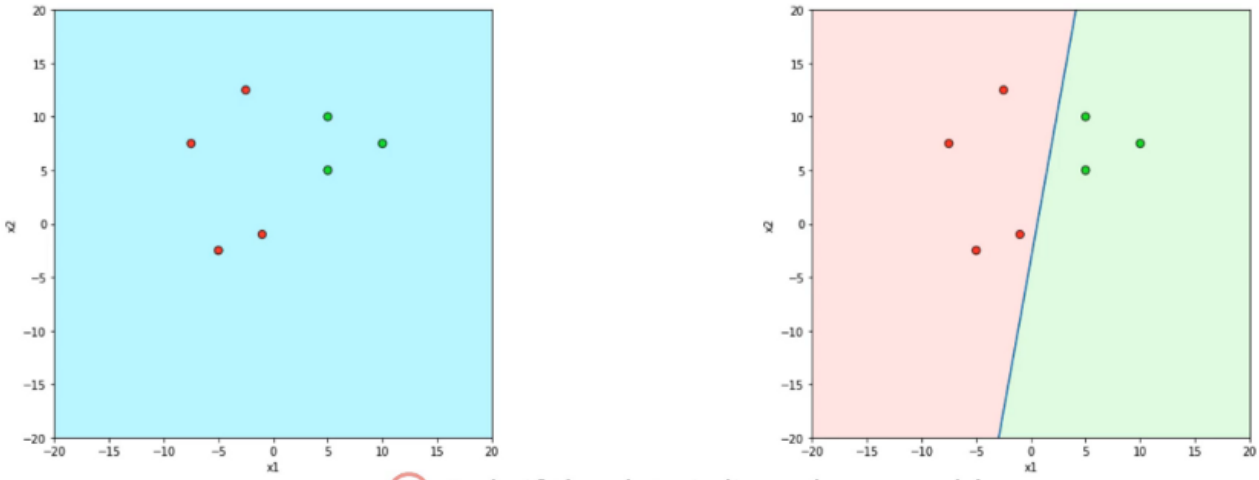
For $\mathbf{x} \in N$ if $\mathbf{w}.\mathbf{x} \geq 0$ then it means that the angle (α) between this **x** and the current **w** is less than 90° (but we want α to be greater than 90°)

What happens to the new angle (α_{new}) when $\mathbf{w_{new}} = \mathbf{w} - \mathbf{x}$

$$
\begin{aligned}
cos(\alpha_{new}) &\propto \mathbf{w_{new}}^T \mathbf{x} \\
&\propto (\mathbf{w} - \mathbf{x})^T \mathbf{x} \\
&\propto \mathbf{w}^T \mathbf{x} - \mathbf{x}^T \mathbf{x} \\
&\propto cos\alpha - \mathbf{x}^T \mathbf{x} \\
cos(\alpha_{new}) &< cos\alpha
\end{aligned}
$$

Will It Always Work?

The Perceptron Learning Algorithm will converge only if the data is linearly separable.



If that is not the case, then the Perceptron model would keep toggling, and make an error sometimes on the positive point, sometimes on the negative point and so on.

And there is this proof which tells that it would always work for linearly separable data.

Definition: Two sets P and N of points in an n-dimensional space are called absolutely linearly separable if $n+1$ real numbers $w_0, w_1, ..., w_n$ exist such that every point $(x_1, x_2, ..., x_n) \in P$ satisfies $\sum_{i=1}^{n} w_i * x_i > w_0$ and every point $(x_1, x_2, ..., x_n) \in N$ satisfies $\sum_{i=1}^{n} w_i * x_i < w_0$

linearly separable the Perceptron Learning algorithm will converge in a finite number of steps.

(c) One Fourth Labs

Perceptron Evaluation:

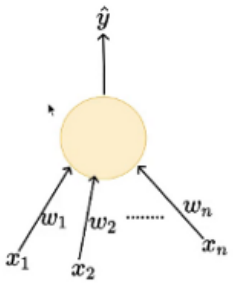
Once the model is ready, we want to evaluate its performance on a test set.

Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (y)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

$$\hat{y} = (\sum_{i=1}^{n} w_i x_i \geq b)$$

$$loss = \sum_i \mathbf{1}_{(y_i != \hat{y}_i)}$$



Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight	0.19	0.63	0.33	1	0.36	0.66	0	0.70	0.48
Screen size	0.64	0.87	0.67	0.88	0.7	0.91	0	1	0.47
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery	0.36	0.51	0.36	1	0.34	0.67	0	0.57	0.43
Price	0.09	0.63	0.41	0.19	0.06	0	0.72	0.94	1
Like (y)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**

Test data

1	0	0	1
0.23	0.34	0.44	0.54
0.74	0.93	0.34	0.42
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Test data

1	0	0	1
0.23	0.34	0.44	0.54
0.74	0.93	0.34	0.42
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$= \frac{3}{4} = 75\%$$

Summary:

Data: We are now dealing with Real inputs(and not boolean inputs)

Task: We are still dealing with Classification

Model:

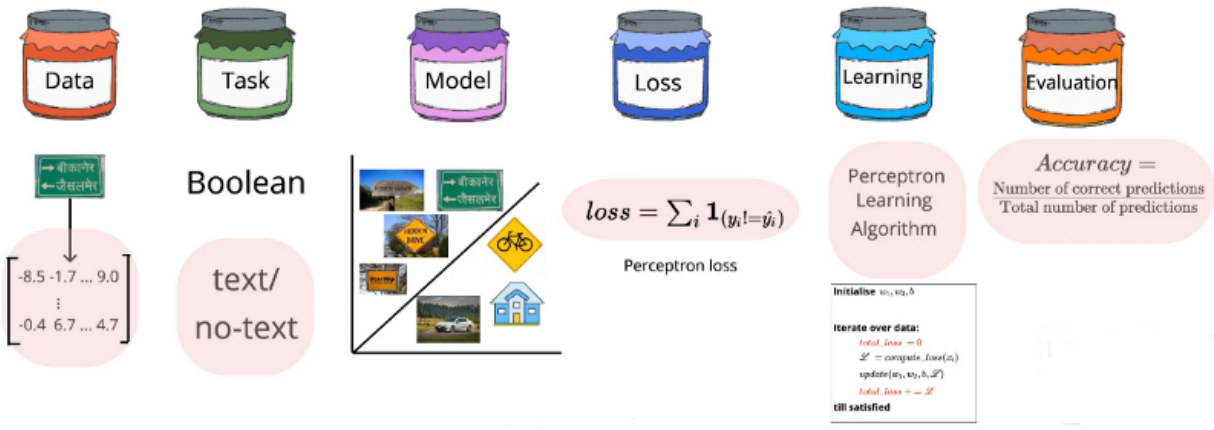
The perceptron model also tries to find a line that separates the positive points from the negative ones, it's just that as opposed to the MP Neuron model, we have more parameters here which mean more flexibility to adjust the line.

Loss: takes on a value of 1 if the true output is different from the predicted output else it is 0.

Learning Algorithm: We keep going over the data, we look at every data point, we compute some loss and based on that we take any action, the action, in this case, is to adjust the parameters. The only limitation of this algorithm is that if the data is not linearly separable, then the algorithm would not converge.

Accuracy: is just the number of correct predictions divided by the total number of predictions.

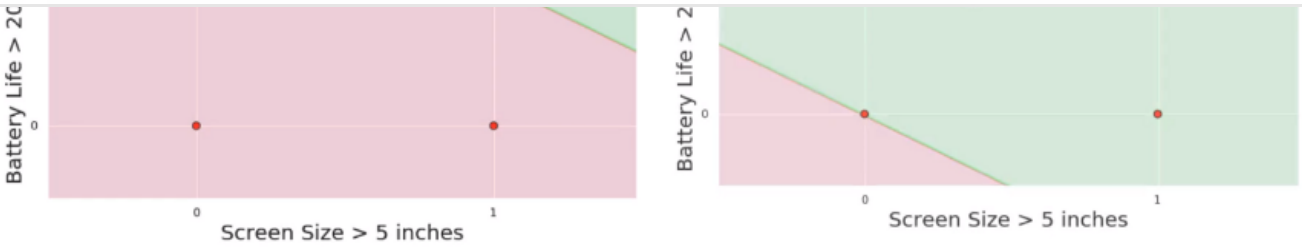
Perceptron model is the simplest model for classification.

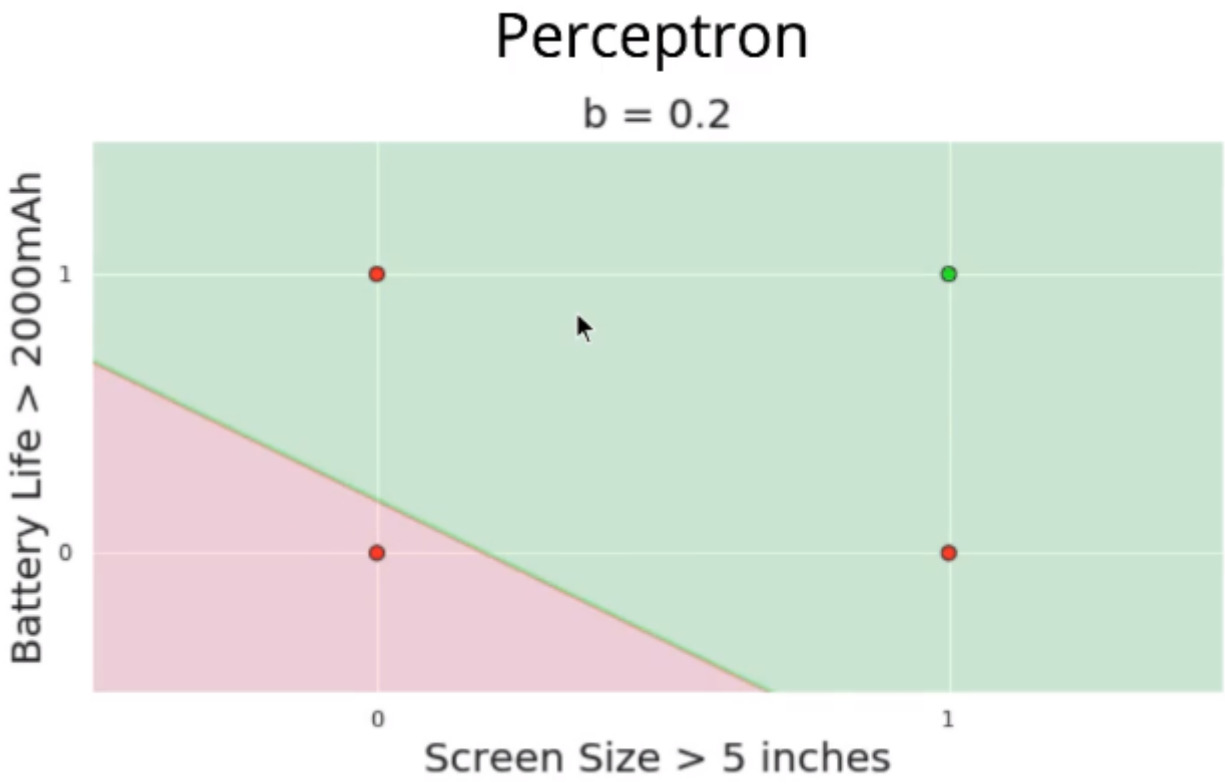


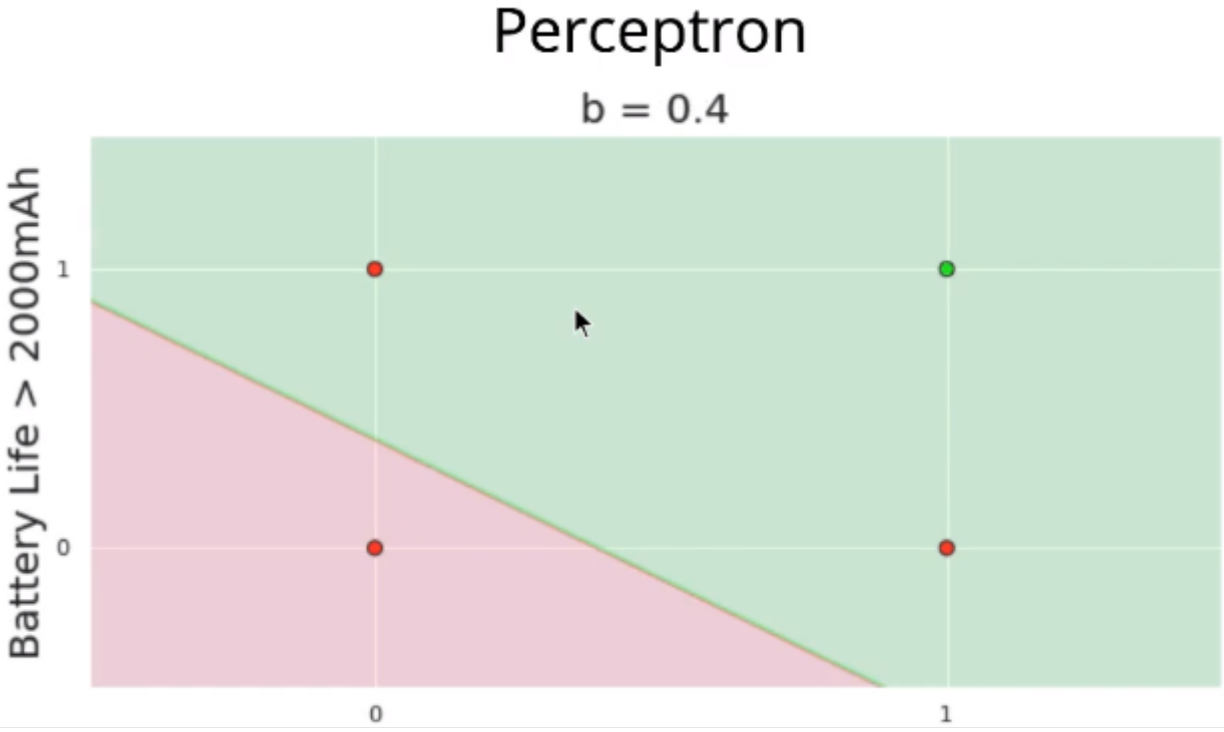
Perceptron Geometrical Interpretation:

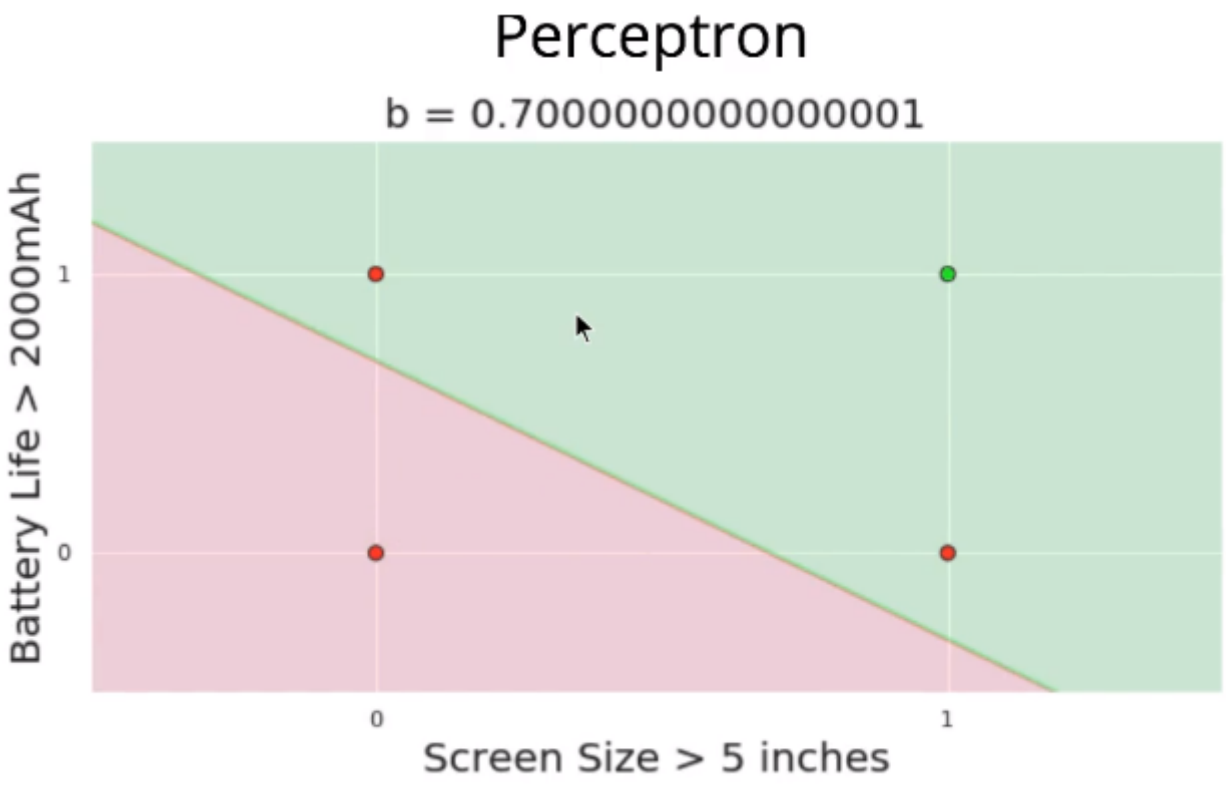
Here the **parameter b can be continuous**, it could be any value that we want, there are no restrictions, it could be any real no. which gives more flexibility to adjust this line to achieve the desired goal to separate positive points from negative points.

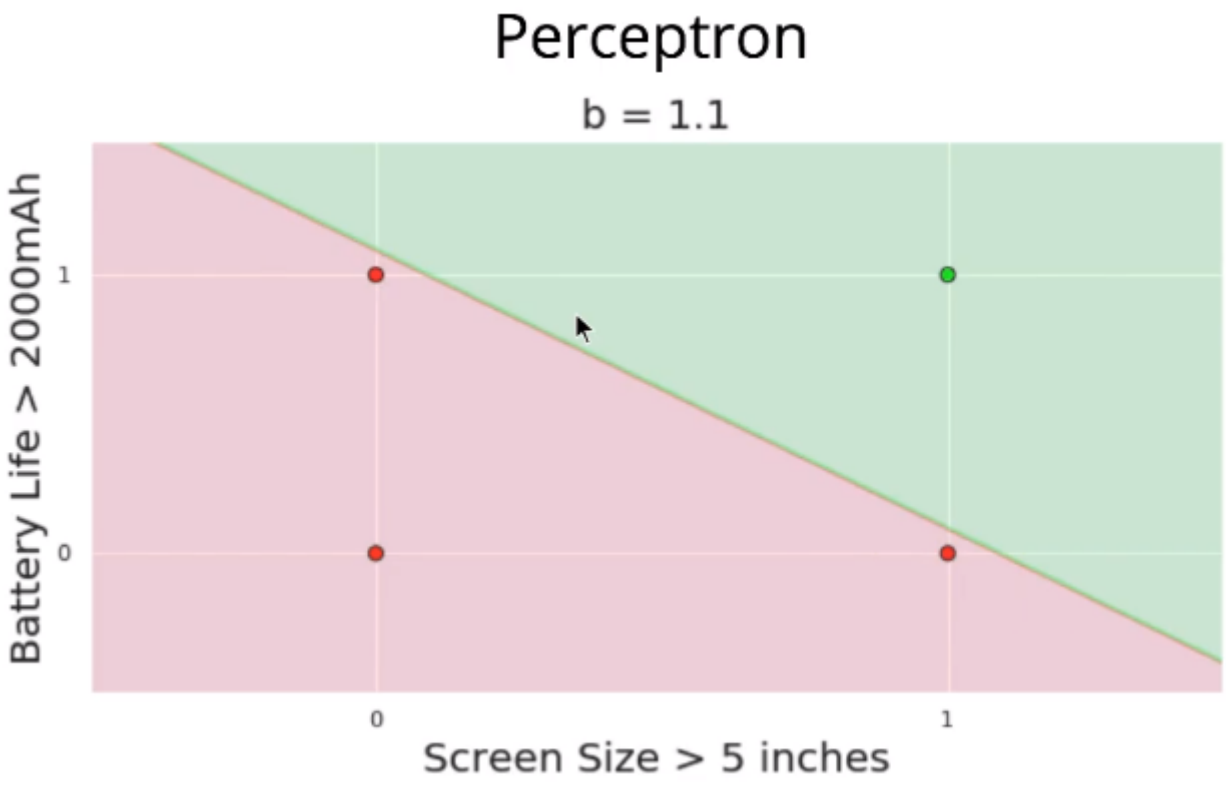
MP neuron Perceptron

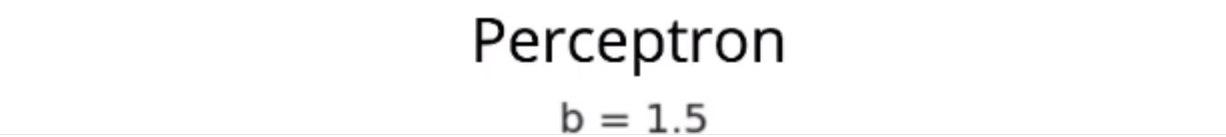

Battery Life > 2
0
0
1
Screen Size > 5 inches
Battery Life > 2
0
0
1
Screen Size > 5 inches

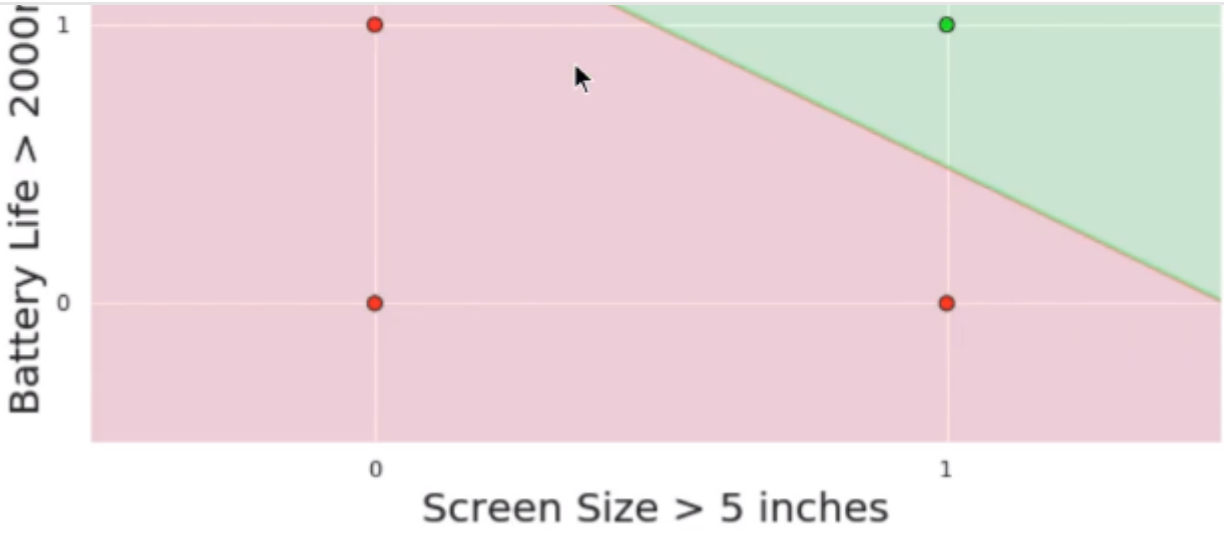

Perceptron
b = 0.2
Battery Life > 2000mAh
1
0
0
1
Screen Size > 5 inches

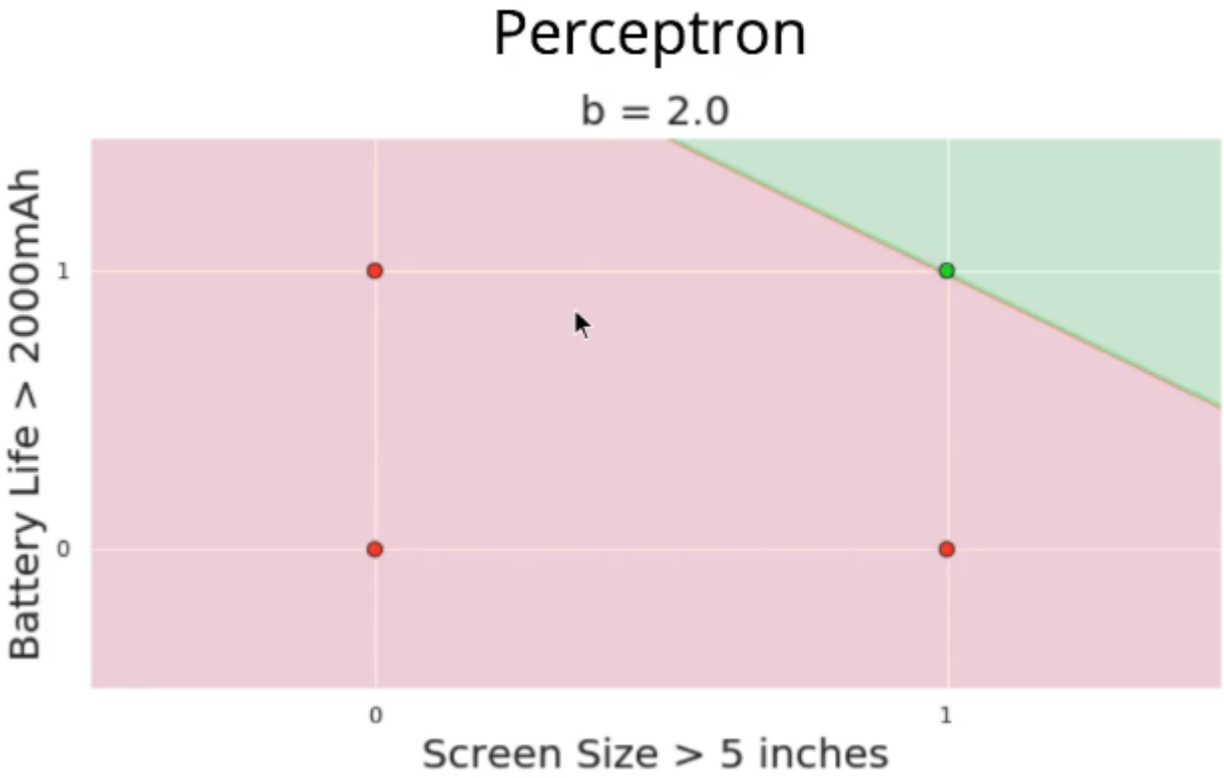

Perceptron
b = 0.4
Battery Life > 2000mAh
1
0
0
1


Perceptron
b = 0.7000000000000001
Battery Life > 2000mAh
1
0
0
1
Screen Size > 5 inches

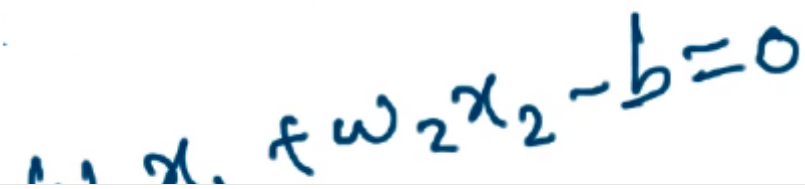

Perceptron
b = 1.1
Battery Life > 2000mAh
1
0
0
1
Screen Size > 5 inches


Perceptron
b = 1.5



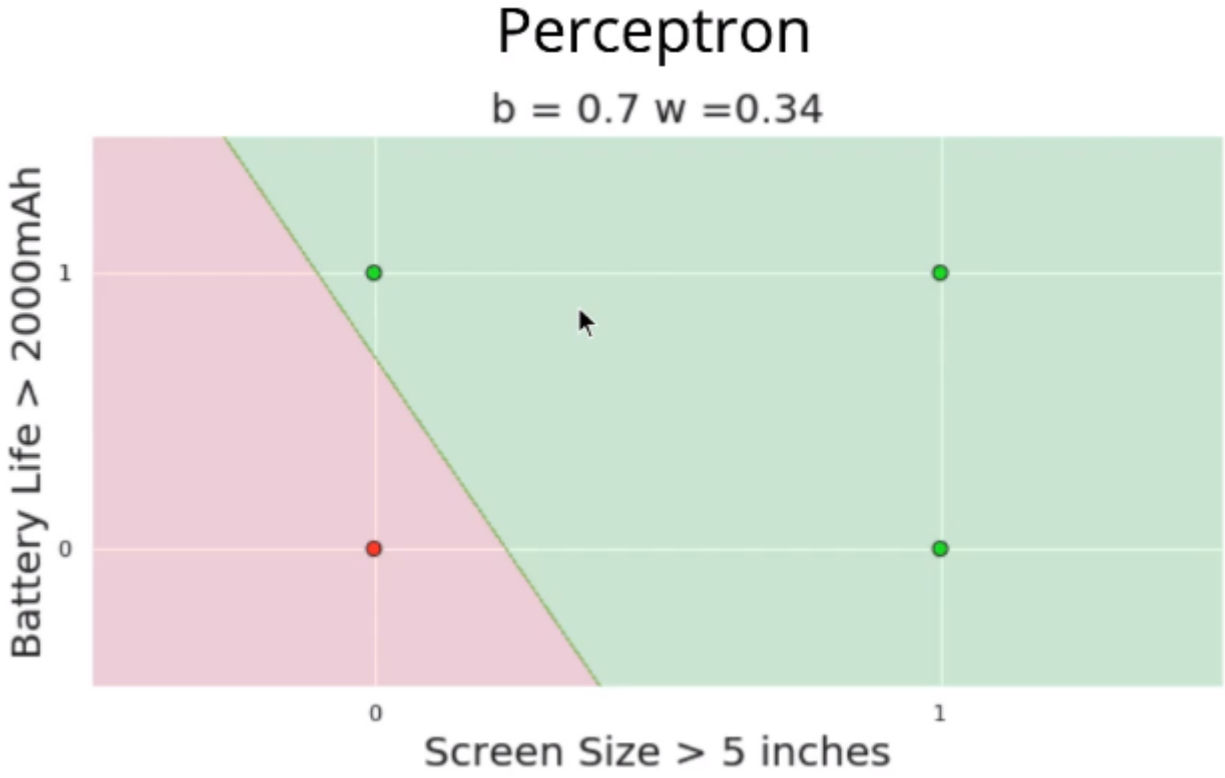


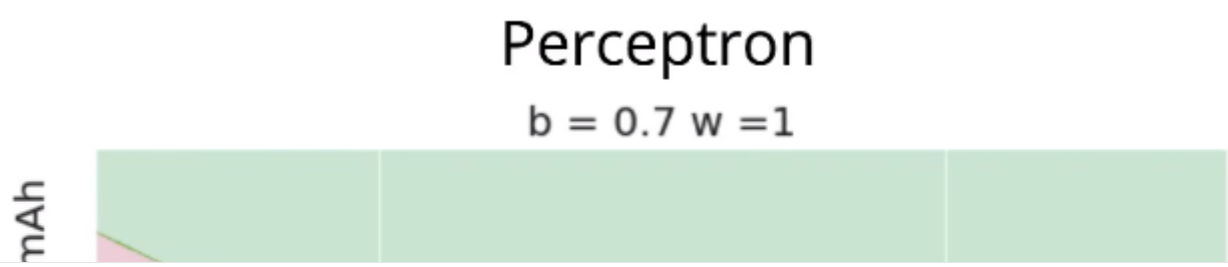
The other point is that this line(which acts as a boundary) has a slope that we can adjust(in Perceptron case) as the value of slope depends on the weights(w1, w2 in 2D), and since we can adjust the weights that mean we can adjust the slope.

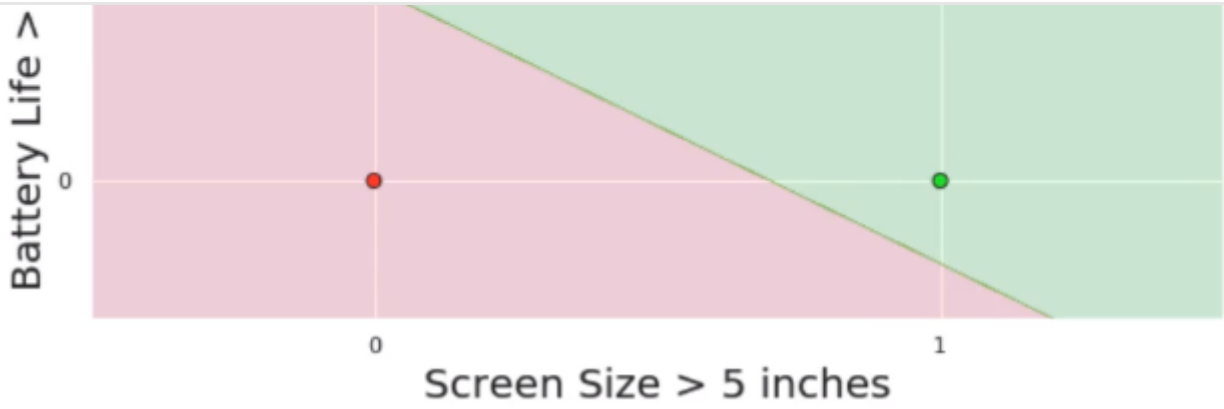


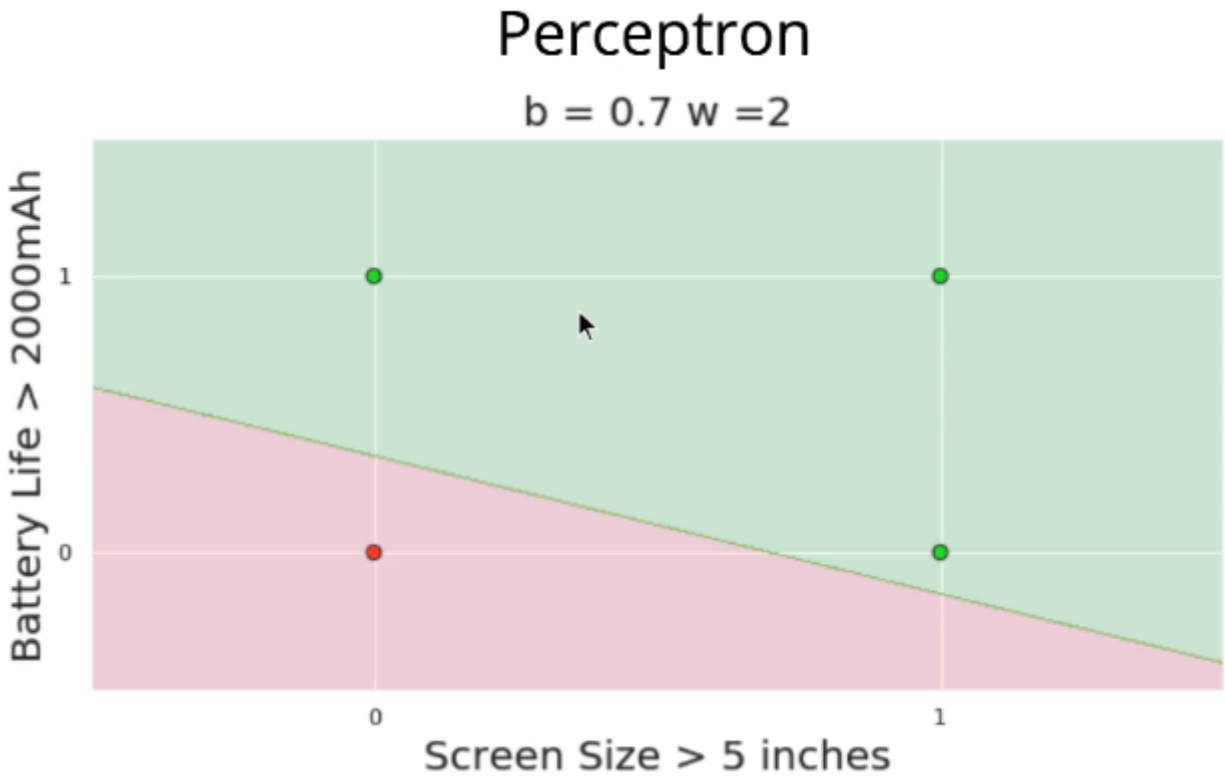
$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{b}{w_2}$$

$$m = -\frac{w_1}{w_2}$$





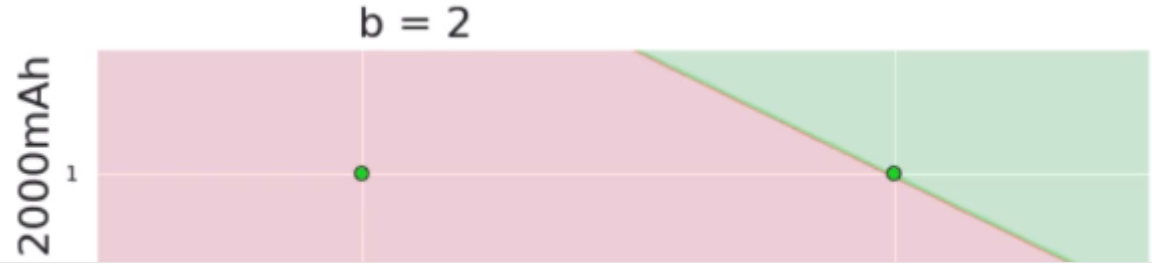


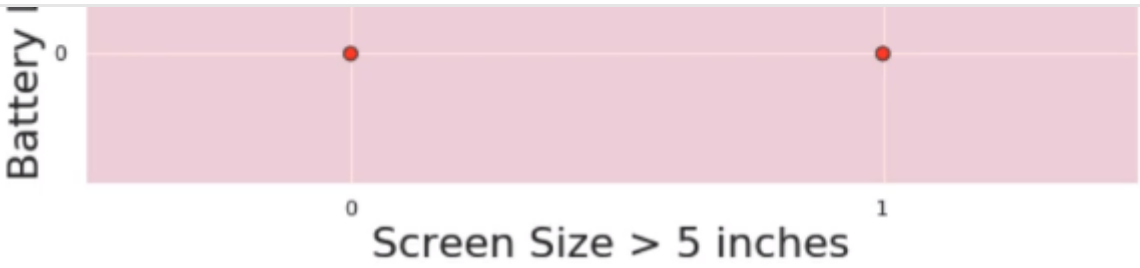


So, **perceptron model has more freedom compared to MP Neuron model.**

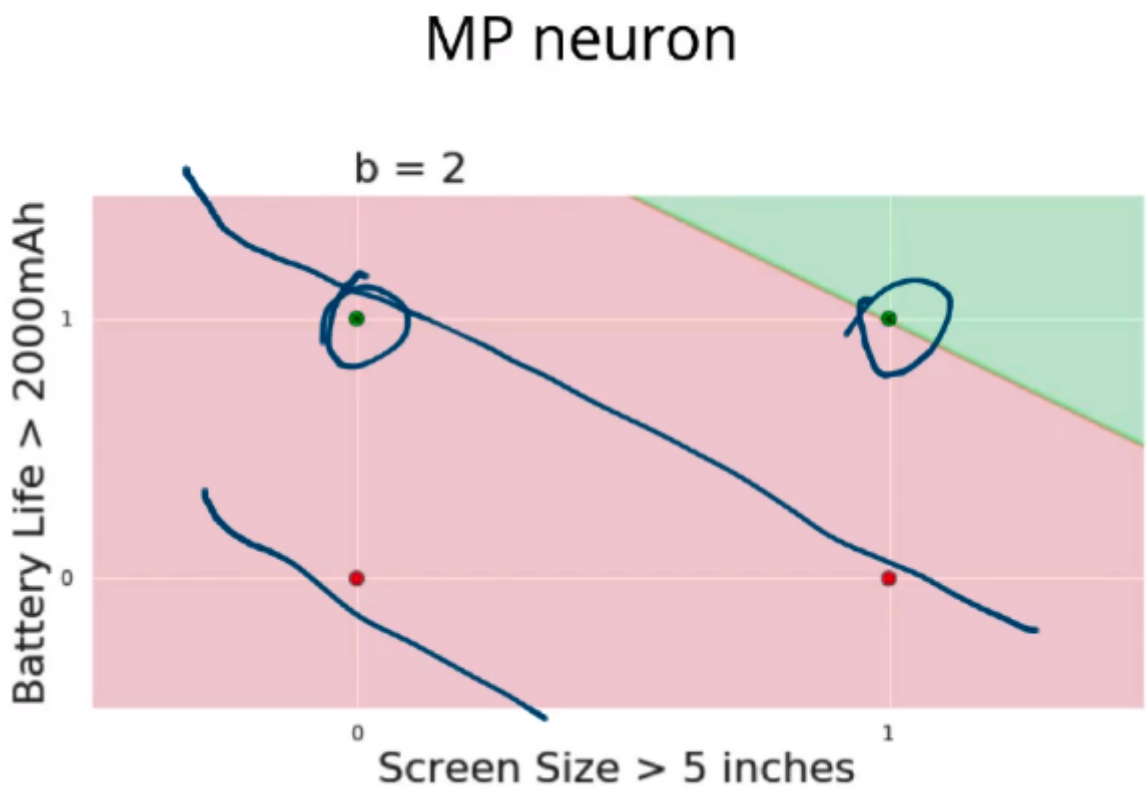
Why is more freedom important?

Let's say the data looks like the below

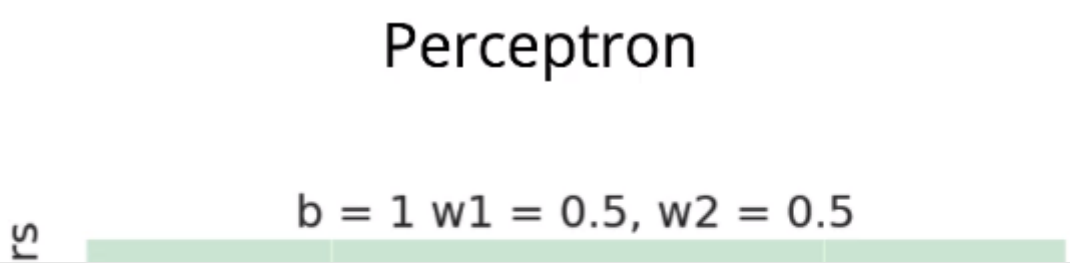


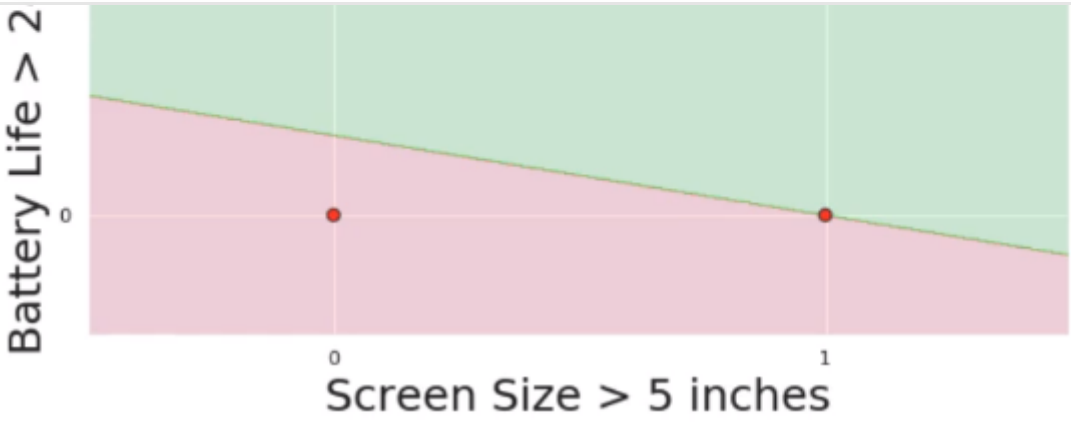


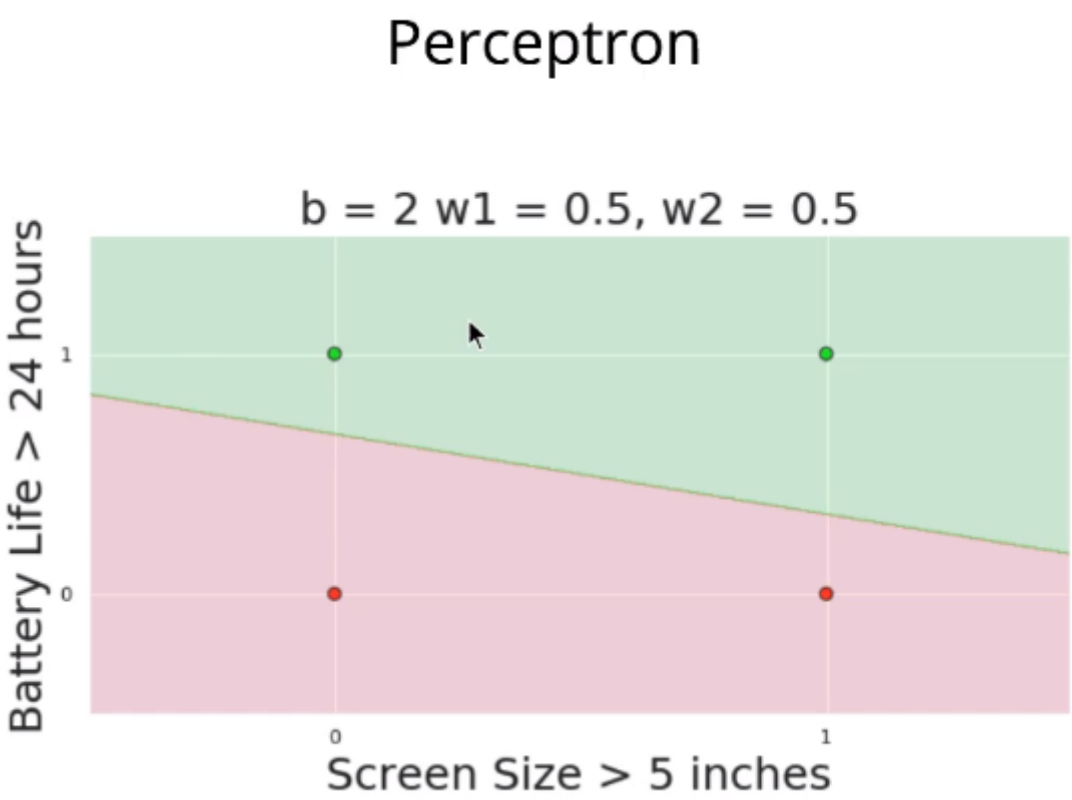
So, for this case, the only lines we could draw(using MP Neuron model) are as in the below image and in this case, we could not have come up with a line which we will separate all the positive points from the negative points:



And because of the flexibility that the Perceptron model has with respect to the slope as well as with the value of the threshold(value where the line touches the x2 axis), using that we are able to get a line which separates the positive points from the negative points.



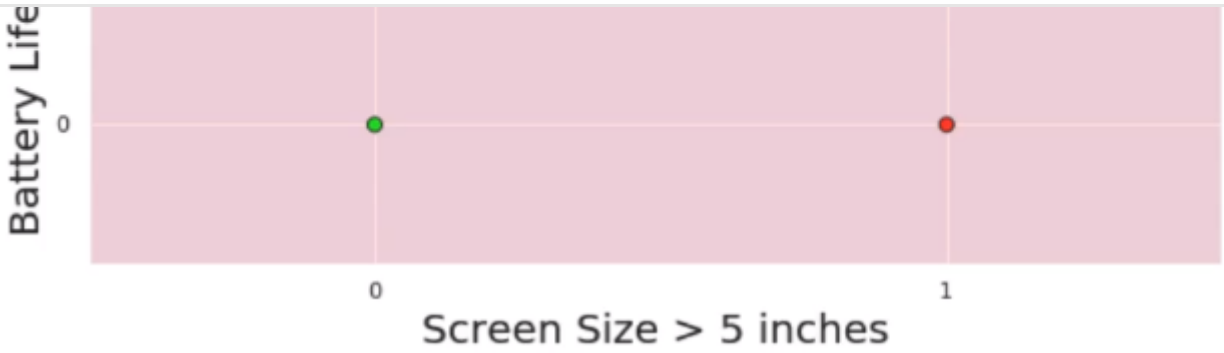




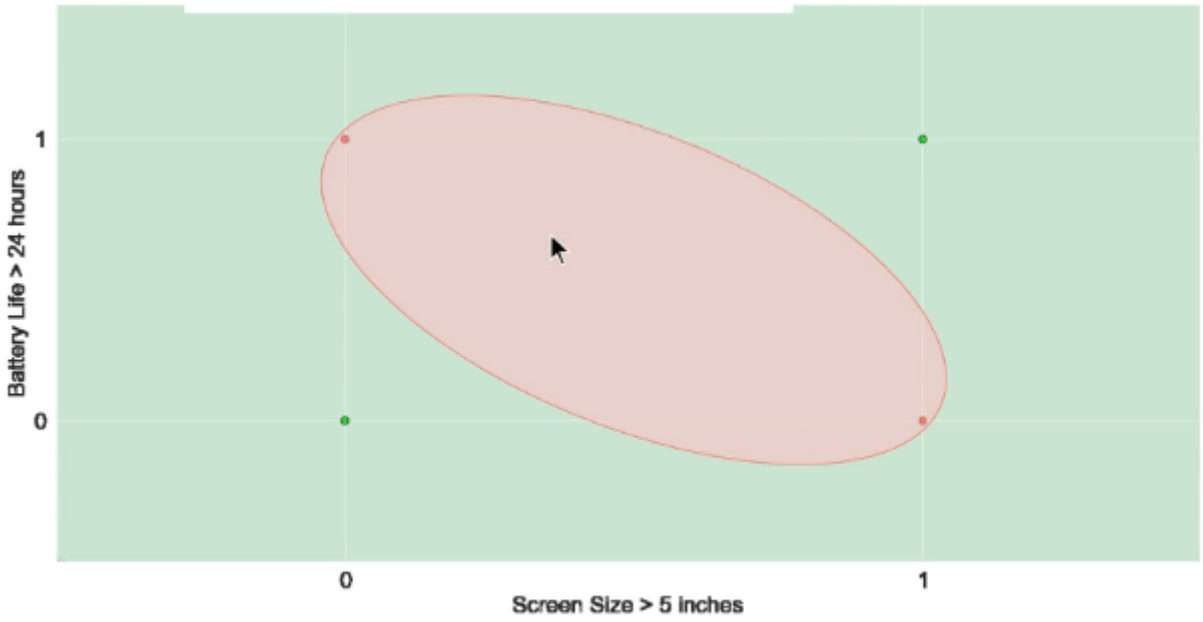
So, **more flexibility leads to dealing with more complex data** also, where separating the positive points from negative points would have been tricky.

Now the question is, **Is this freedom enough?**

Let's consider the below case where the data points are as depicted below:



Here, in this case, no matter how we draw the line, we can not separate the positive points from the negative points. This issue still holds for Perceptron model because the kind of model that we need to deal with this data is something of this sort:



In other words, we can say that the **Perceptron can only deal with Linearly Separable data**. So, **our ideal model requires more freedom so that it can deal with the data that is not linearly separable**.

Machine Learning Deep Learning Perceptron Perceptron Algorithm Artificial Intelligence